MIS Checklist

Spencer Smith

November 11, 2025

If a team adopts a documentation approach that does not use an MIS, then some of the items in this checklist will not apply.

- Follows writing checklist (full checklist provided in a separate document)
  - ☐ LaTeX points
  - ☐ Structure
  - ☐ Spelling, grammar, attention to detail
  - ☐ Avoid low information content phrases
  - ☐ Writing style
  - ☐ Hyperlinks should be done properly (`\ref`)
  - ☐ Every module's specification starts on a new page
- MIS Module Classifications
  - ☐ Types that only hold data (records) are modelled as exported types. For instance, the StdntAllocTypes module in A2: https://gitlab.cas.mcmaster.ca/smiths/se2aa4_cs2me3/blob/master/Assignments/A2/A2.pdf)
  - ☐ Types that have data (state) and behaviour are modelled as ADTs. The MIS should use the keyword **Template**. An example is the BoardT ADT given at https://gitlab.cas.mcmaster.ca/smiths/se2aa4_cs2me3/blob/master/Assignments/A3/A3Soln/A3P1_Spec.pdf

- ☐ Abstract objects are used when there is only one instance. There is state and behaviour. This most often comes up for "global" reader and writer modules. For instance, a module that does logging. Abstract objects do NOT use the word Template in the main header. An example is given in the SALst module of A2: https://gitlab.cas.mcmaster.ca/smiths/se2aa4_cs2me3/blob/master/Assignments/A2/A2.pdf
- ☐ Library modules are used when there is only behaviour, no state. They are defined as Modules, but State Variables and Environment Variable fields say "None."
- ☐ If the module's MIS can be parameterized by type, then the keyword **Generic** is used. Generic modules are usually also Template modules, but not necessarily. An example is given in the Generic Stack Module (Stack(T)) given in A3: https://gitlab.cas.mcmaster.ca/smiths/se2aa4_cs2me3/blob/master/Assignments/A3/A3Soln/A3P1_Spec.pdf
- ☐ Abstract objects will have some kind of initialization method
- ☐ Abstract objects will have an assumptions that programmers will initialize first, or a state variable that is set from False to True when the Abstract object is initialized - this state variable then needs to be checked for each access program call

- MIS and Mathematical syntax
  - ☐ Exported constants are "hard-coded" literal values, not variables. Constants are values that are known at specification (and therefore compile) time. Explicit constant values are provided in the MIS, not left to be filled in later. (They can be changed later, but specific values should be given.)
  - ☐ Operations do not mix incorrect types. For instance, a character is not added to an integer, an integer is not "anded" with a Boolean, etc.
  - ☐ Our modified Hoffmann and Strooper notation is used, or any new notation is clearly defined.
  - ☐ Notation is consistent throughout the document.

- ☐ All arguments to a method are used in the specification of the method in the semantics section
- ☐ All local functions are used somewhere in the modules's specification.
- ☐ The uses relation is for the modules that need to be referenced to completely define the specification. If parts of the exported interface from those modules are not needed, then the modules aren't listed under Uses.

- MIS Semantics for each module
  - ☐ Each access program does something – either an output, or a state transition
  - ☐ Access programs either change the state of something, or have an output. Only rarely should an access program do both (as it does for the constructor in an ADT.)
  - ☐ If there is an entry in the state transition, then the state of something changes. The state change might be the local state variables, the state variables for another module, or an environment variable.
  - ☐ Outputs use $out := ...$
  - ☐ Exceptions use $exc := ...$
  - ☐ If the state invariant is satisfied before an access program call, it will remain satisfied after the call
  - ☐ State invariant is initially satisfied
  - ☐ Local functions make the specification easier to read (there is no requirement that the local functions will actually be implemented in code)
  - ☐ Modules that deal with files, the keyboard, or the screen, have environment variables to represent these respective entities
  - ☐ Symbols are from SRS - not yet translated to code names (that is use θ, not `theta`)

- MIS Quality inspection for each module
  - ☐ Consistent

- ☐ Essential
- ☐ General
- ☐ Implementation independent
- ☐ Minimal
- ☐ High cohesion
- ☐ Opaque (information hiding)
- ☐ Correct level of abstraction (mentioning JSON or using Python's init syntax is too low-level)

- MIS Completeness
  - ☐ All types introduced in the spec are defined somewhere
  - ☐ All modules in MG are in the MIS
  - ☐ All required sections of the template are present for all modules